Design and Performance Report

On

Parallel Word Document Index

by

SHIVANKIT GAIND	2015A7PS0076P
ROHIT GHIVDONDE	2015A7PS0077P

For the partial fulfilment of the
course on

Parallel Computing

Submitted to
Prof. Shan Sundar Balasubramaniam

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

February, 2018

Design

The distributed index construction method we have used in this problem is an application of MapReduce , a general architecture for distributed computing.

1. In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of key-value pairs.

2. The map phase split up the computing job into chunks that standard machines can process in a short time. In our case, the **mapping phase is the Local Index Construction** from documents available at each node. **We assume that the input data is already distributed on different nodes,** so there is no need to divide data among processes. So, we run **one process on each node** to **compute local index on that node**.

3. We assume that the **local index on a particular node can fit into memory** (RAM), so we used the **UNORDERED MAP** data structure from C++ to store the inverted index on each node. The map stores a list (in our case, **vector** in C++) of pairs of **Document Id and Term Frequency** (in that document) corresponding to each **Term**.

4. After Local Index is created, the posting list corresponding to each document will be **sorted in decreasing order of term frequencies** (We could have done sorted insert as well at the time of insertion only or could have used a heap, but the time complexity would have remained same or worse).

5. In our Case, the **Local Index is also splitted into partitions equal to total number of nodes**, hence, the posting list for any word will be present in the Map corresponding to the partition the word belongs to. This will help in reducing phase where each merger/reducer will pick-up the maps belonging to one partition only from every node.

6. Which partition a particular term belongs to, depends upon the **number of partitions and the partitioning scheme.** In our implementation, we are partitioning the words on the basis of first letter. A better scheme could also be partitioning by hashing the words into a partition index, but here, we have kept it simple.

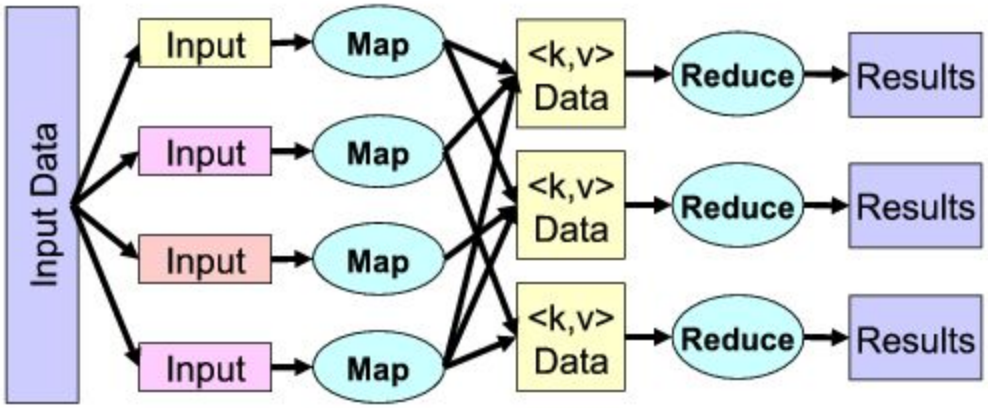
7. After the Local index is created on every node, i.e the mapping phase is over, **the same nodes now play the roles of mergers/reducers**.

8. Since, every Local Index is splitted into partitions equal to total number of nodes, so i^{th} reducer will pick up the maps from i^{th} partition of every node.

9. Sending appropriate partition to the corresponding reducer is where **communication** comes into play. For that, a library(**cereal**) is used to serialize the maps into binary format(**stringstream based serialization**) and these **binary strings** are then sent by each process to other processes (according to the partition the corresponding process is handling) using **MPI_Alltoall** call. In case the **data becomes too large** to be sent using MPI calls, the communication mode is changed to the **filestream based serialization** where the **serialized maps are stored in files in NFS** and the reducers read the maps from the files corresponding to their partition **during the reducing phase**.

10. Since every reducer has posting lists corresponding to a subset of words, where **no words are common between reducers**, each reducer will merge the received maps and store the index locally.

11. In this way, a **Distributed Global Index** will be formed. The query for a word can now be sent to an appropriate node, by calculating the partition the word belongs to.



Performance Evaluation

The Description of the Performance Measurements is as follows:

1. **‘p’ corresponds to number of nodes, where one process runs on every node.**
2. The **time measurements** are done in **seconds.**
3. The datasizes (100,200,400,800 and 1600) are in **MB**.
4. The time for local index creation, global index creation(includes) and total time are summarized in the table.

IMPORTANT:

For every cell in the table, p=x and datasize = y means, that x processes were running (one on each of the x nodes) and each process worked on y sized data locally.

Hence, for column of 200, for p=1, total data worked upon is 200 MB whereas for p=2, total data worked upon = 400MB (200MB by each process). Similarly, for 1600MB, p = 8 means, total data worked upon is 1600*8MB.

Note:

1. Local index formation time is the time required to read the files and form the local indices.
2. Global index formation time includes both communication time and the local indices merging time.
3. Total time is the sum of global and local index formation time.

LOCAL INDEX	100 MB/node	200 MB/node	400 MB/node	800 MB/node	1600 MB/node
p=1	4.798995	10.585609	19.099528	38.19541	78.459706
p=2	4.931753	9.400376	19.401346	38.882512	78.352298
p=4	4.801977	9.663481	18.94347	37.97544	77.785117
p=8	4.764668	9.511744	18.901517	38.32558	78.236706
TOTAL TIME	100 MB/node	200 MB/node	400 MB/node	800 MB/node	1600 MB/node
p=1	5.062929	10.9434	19.56493	38.921425	80.133514
p=2	5.300039	9.794719	19.945449	39.890985	80.493486
p=4	5.171345	10.315004	19.872523	40.149169	82.681618
p=8	5.419858	10.375699	20.166158	43.48827	86.195253

GLOBAL INDEX	100 MB/node	200 MB/node	400 MB/node	800 MB/node	1600 MB/node
p=1	0.263934	0.357791	0.465402	0.726015	1.673808
p=2	0.368286	0.394343	0.544103	1.008473	2.141188
p=4	0.369368	0.651523	0.929053	2.173729	4.896501
p=8	0.65519	0.863955	1.264641	5.16269	7.958547

Performance improvement for same data when splitted across nodes:

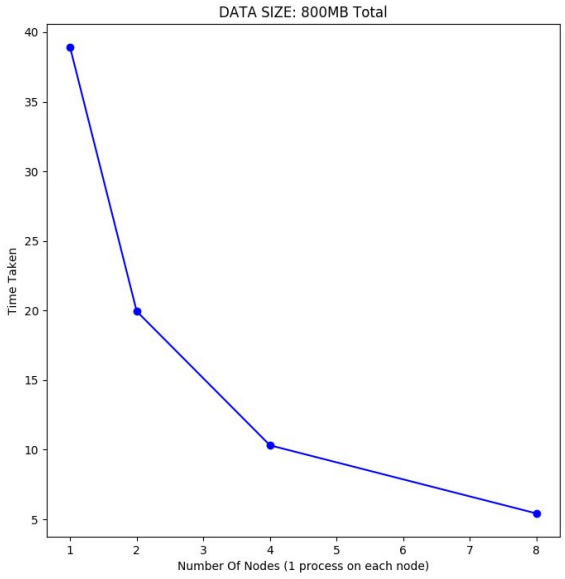
The table given below shows for performances improves when the number of processes working on same amount of data increases.

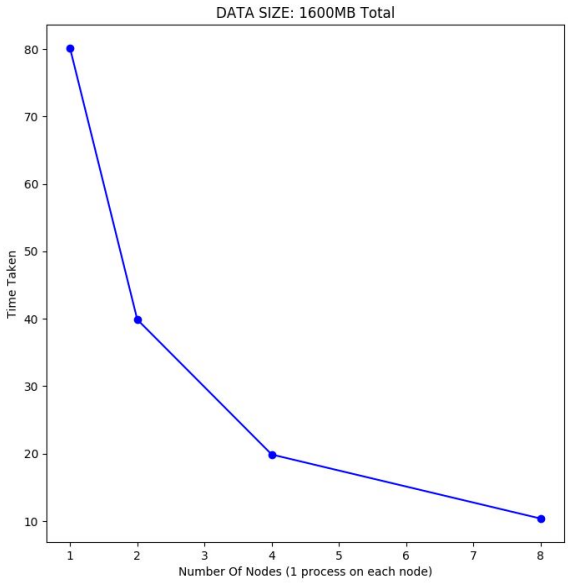
IMPORTANT:

Hence for a column of 800MB, p = 1 means all 800MB data was worked upon by one process on one node. p=2 means 800MB data was splitted across 2 nodes (400MB each) and one process worked on each split. Similarly, p = 8 means that each of the 8 process worked on a split of 100MB at each node for total data of 800MB.

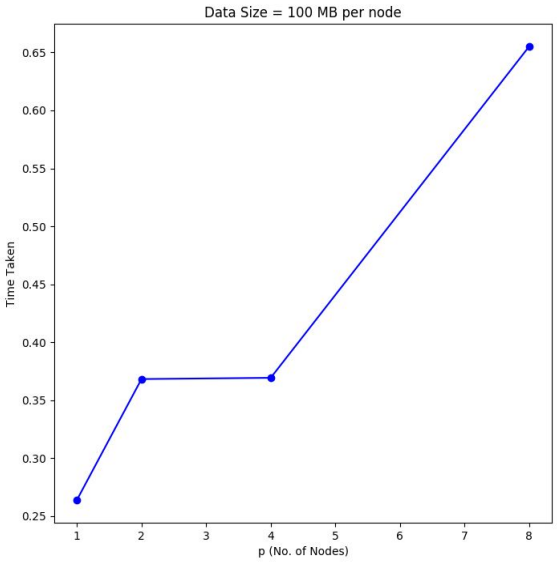
TOTAL	800 MB Total	1600 MB Total	3200 MB Total	6400 MB Total
p=1	38.921425	80.133514	-	-
p=2	19.945449	39.890985	80.493486	-
p=4	10.315004	19.872523	40.149169	82.681618
p=8	5.419858	10.375699	20.166158	43.48827

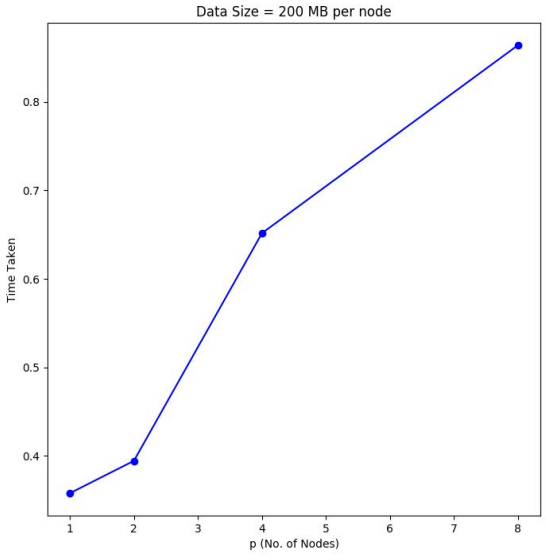
A. The plots showing the improvements are as follows:

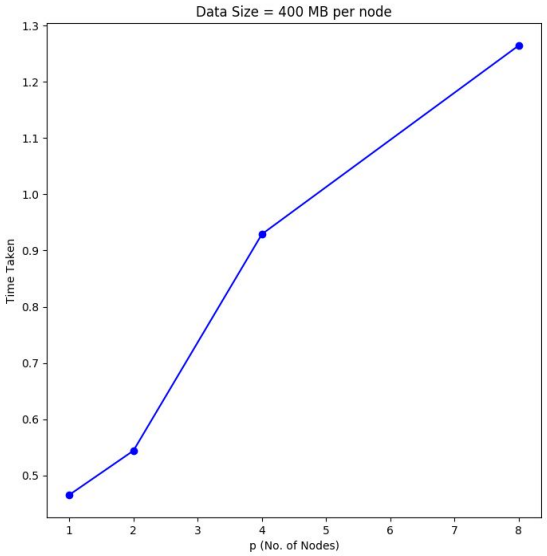


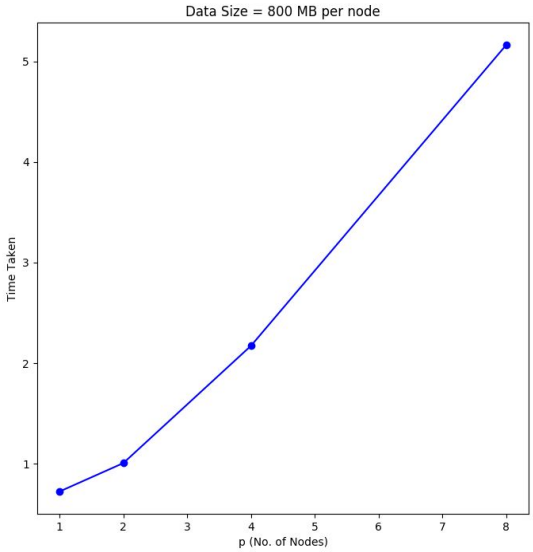


B. The plots showing how global index calculation time increases with the increase in the number of nodes(keeping data per node same) are as follows:



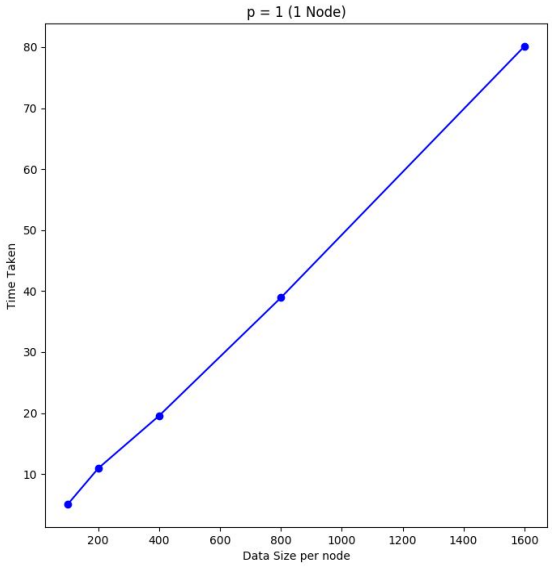


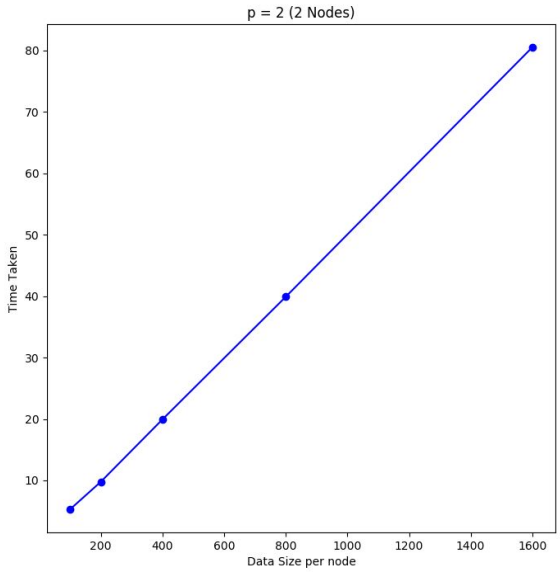


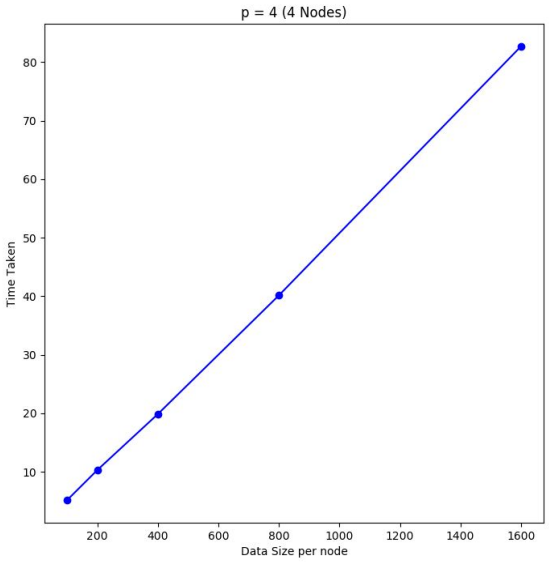


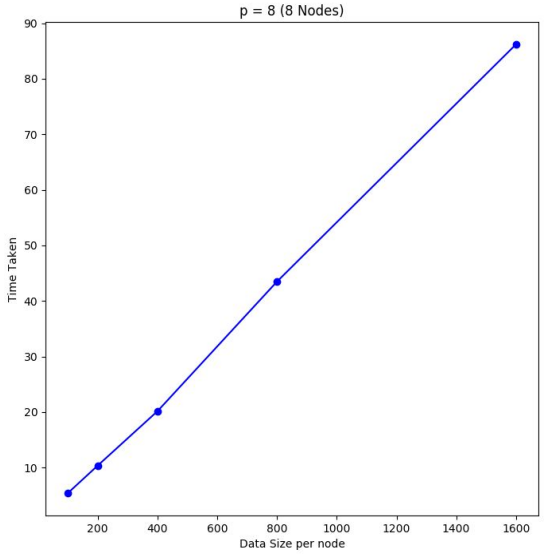
The plots clearly indicate that with the increase in the number of nodes, time required for global index formation **increases** due to an **increase in the communication** between the nodes.

C. The plots showing how time increases with increase in the amount of data per node are as follows:






p = 4 (4 Nodes)
Time Taken
Data Size per node
10
20
30
40
50
60
70
80
200
400
600
800
1000
1200
1400
1600


p = 8 (8 Nodes)
Time Taken
Data Size per node
10
20
30
40
50
60
70
80
90
200
400
600
800
1000
1200
1400
1600